

Rangkuman pertemuan 1-2 pengenalan dan studi kasus

Pemrograman berorientasi objek (Object Oriented Programming atau disingkat OOP) adalah paradigma pemrograman yang berorientasikan kepada objek yang merupakan suatu metode dalam pembuatan program, dengan tujuan untuk menyelesaikan kompleksnya berbagai masalah program yang terus meningkat. Pemrograman berorientasi objek ditemukan pada Tahun 1960, dimana berawal dari suatu pembuatan program yang terstruktur (structured programming). Metode ini dikembangkan dari bahasa C dan Pascal.

Perbedaan pemrograman berbasis objek dengan pemrograman terstruktur:

1. Pemrograman berbasis objek:

- Menggabungkan fungsi dan data dalam kelas – kelas atau objek – objek.
- Memiliki ciri Encapsulation (pengemasan), Inheritance (penurunan sifat) dan Polymorphism (perbedaan bentuk dan perilaku).
- Struktur program ringkas.
- Kode program sangat re-usable.
- Efektif digunakan untuk menyelesaikan masalah besar, karena OOP terdiri dari class-class yang memisahkan setiap kode program menjadi kelompok - kelompok kecil.
- Sulit diawal (karena harus membuat class).
- Eksekusi lebih cepat karena dieksekusi bersamaan.

2. Pemrograman terstruktur:

- Memecah program dalam fungsi dan data.
- Memiliki ciri Sequence (berurutan), Selection (pemilihan) dan Repetition (perulangan).
- Struktur program rumit karena berupa urutan proses dan fungsi-fungsi.
- Re-use kode program kurang.
- Efektif digunakan untuk menyelesaikan masalah kecil.
- Mudah diawal
- Eksekusi lebih lambat

Class adalah cetak biru atau blueprint dari object. Class digunakan hanya untuk membuat kerangka dasar.

```
// Deklarasi Class
public class Kendaraan{
    // Ini Adalah Class, semua konstruktor, variabel, method berada disini
}
```

Suatu objek didefinisikan berdasarkan namanya atau menggunakan kata benda, objek juga memiliki atribut dan metode. Pada dasarnya ada dua karakteristik yang utama pada sebuah objek, yaitu :

1. Setiap objek memiliki atribut sebagai status yang kemudian akan disebut sebagai **state**.
2. Setiap objek memiliki tingkah laku yang kemudian akan disebut sebagai **behavior**

Dalam pengembangan perangkat lunak berorientasi objek, objek dalam perangkat lunak akan menyimpan **state-nya dalam variable** dan menyimpan informasi **tingkah laku (behaviour) dalam method atau fungsi-fungsi/prosedur**. Untuk membuat objek, bisa menggunakan kata kunci new yang digunakan untuk membuat objek baru, yaitu: mendeklarasikan variable, membuat objek baru (Instansiasi) dan pemanggilan konstruktor.

```
public class Kendaraan{
    // Konstruktor Dengan Parameter
    public Kendaraan(String nama){
        System.out.println("Nama Kendaraannya Adalah " + nama);
    }

    public static void main(String[] args){
        // Perintah untuk membuat objek jenis
        Kendaraan jenis = new Kendaraan("Pesawat Terbang");
    }
}

// Output = Nama Kendaraannya Adalah Pesawat Terbang
```

Studi Kasus

Buatlah program menggunakan Java dengan kasus seperti berikut:

Seorang pedagang sate setiap harinya menjual dan menerima pesanan dari beberapa pembeli, harga pertusuknya adalah Rp.950, suatu hari ada pembeli yang memesan 115 tusuk. Jadi berapa harga sate 115 tusuk yang dibayarkan oleh pembeli?

Main.java	Output
<pre>1 import java.util.Scanner; 2 3 public class Hargasate { 4 public static void main(String[] args) { 5 Scanner input = new Scanner(System.in); 6 7 System.out.print("Nama Barang: "); 8 String namaBarang = input.nextLine(); 9 10 System.out.print("Harga Barang: Rp. "); 11 int hargaBarang = input.nextInt(); 12 13 System.out.print("Jumlah Barang: "); 14 int jumlahBarang = input.nextInt(); 15 16 int totalHarga = hargaBarang * jumlahBarang; 17 18 System.out.println("Total Harga: Rp. " + totalHarga); 19 } 20 }</pre>	<pre>java -cp /tmp/PutKjxkavd/Hargasate Nama Barang:sate Harga Barang: Rp. 950 Jumlah Barang: 115 Total Harga: Rp. 109250 === Code Execution Successful ===</pre>

PERTEMUAN KE 3 - 6 – Inheritance, Encapsulation dan Polymorphism

Pengertian dasar inheritance:

Pewarisan merupakan sebuah bentuk “penggunaan kembali” (*reusability*); dimana **class** baru dibuat dari **class** yang pernah ada yang (biasanya) ditambah fasilitasnya. Pewarisan tunggal (*single inheritance*) merupakan pewarisan dari satu **class** pokok (induk). Pewarisan ganda (*multiple inheritance*). Java tidak mendukung *multiple inheritance*. Class yang mewariskan disebut dengan **superclass / parent class / base class**, sedangkan class yang mewarisi (class yang baru) disebut dengan **subclass / child class / derived class**. Untuk menerapkan inheritance, gunakan statement “extends”. Keyword “**super**” digunakan oleh subclass untuk memanggil constructor, atribut dan method yang ada pada superclass-nya.

Contoh untuk memanggil constructor milik superclass-nya;

```
super()  
super(parameter)
```

Contoh untuk memanggil atribut dan method milik superclass-nya;

```
super.namaAtribut  
super.namaMethod(parameter)
```

Pengertian dasar Encapsulation:

Enkapsulasi (*encapsulation*) merupakan cara untuk melindungi property (atribut) / method tertentu dari sebuah kelas agar tidak sembarangan diakses dan dimodifikasi oleh suatu bagian program. Cara untuk melindungi data yaitu dengan menggunakan access modifiers (hak akses). Ada 4 hak akses yang tersedia, yaitu default, public, protected, private.

Accessors

No	Modifier	Pada class dan interface	Pada method dan variabel
1	Default (tidak ada modifier)	Dapat diakses oleh yang sepaket	Diwarisi oleh subkelas dipaket yang sama, dapat diakses oleh method-method yang sepaket
2	Public	Dapat diakses dimanapun	Diwarisi oleh subkelasnya, dapat diakses dimanapun
3	Protected	Tidak bisa diterapkan	Diwarisi oleh subkelasnya, dapat diakses oleh method-method yang sepaket
4	private	Tidak bisa diterapkan	Tidak dapat diakses dimanapun kecuali oleh method-method yang ada dalam kelas itu sendiri

Aksesabilitas	public	private	protected	default
Dari kelas yang sama	Ya	Ya	Ya	Ya
Dari sembarang kelas dalam paket yang sama	Ya	Tidak	Ya	Ya
Dari sembarang kelas di luar paket	Ya	Tidak	Tidak	Tidak
Dari subkelas dalam paket yang sama	Ya	Tidak	Ya	Ya
Dari subkelas di luar paket	Ya	Tidak	Ya	Tidak

Overloading adalah diperbolehkannya dalam sebuah class memiliki lebih dari satu nama function/method yang sama tetapi memiliki parameter/argument yang berbeda.

```

1 public class Overloading {
2     public void Tampil(){
3         System.out.println("I love Java");
4     }
5     public void Tampil(int i){
6         System.out.println("Method dengan 1 parameter = "+i);
7     }
8     public void Tampil(int i, int j){
9         System.out.println("Method dengan 2 parameter = "+i+" & "+j);
10    }
11    public void Tampil(String str){
12        System.out.println(str);
13    }
14
15    public static void main(String a[]){
16        Overloading objek = new Overloading();
17        objek.Tampil();
18        objek.Tampil(8);
19        objek.Tampil(6,7);
20        objek.Tampil("Hello world");
21    }
22 }

```

Overriding method adalah kemampuan dari subclass untuk memodifikasi method dari superclass-nya, yaitu dengan cara menumpuk (mendefinisikan kembali) method superclass-nya.

Studi Kasus:

Mari kita asumsikan bahwa kita memiliki kelas “Bird” dan kita sedang membuat daftar burung. Mari kita pahami konsep OOP yang digunakan dalam pembuatan kelas “Bird” ini.

INHERITANCE

Untuk setiap burung, ada satu set properti yang telah ditetapkan yang umum untuk semua burung dan ada satu set properti yang khusus untuk burung tertentu. Oleh karena itu, secara intuitif, kita dapat mengatakan bahwa semua burung mewarisi ciri-ciri umum seperti sayap, kaki, mata, dll. Oleh karena itu, dalam cara berorientasi objek untuk merepresentasikan burung, pertama-tama kita mendeklarasikan kelas “Bird” dengan seperangkat properti yang umum untuk semua burung. Dengan melakukan ini, kita dapat menghindari menyatakan sifat-sifat umum ini di setiap burung yang kita buat.

```

// Implementing the bird class
public class Bird {

    // Few properties which
    // define the bird
    String animal = "All Birds";
    String color;
    int legs;

    // Few operations which the
    // bird performs
    public void eat()
    {
        System.out.println(
            "This bird has eaten");
    }

    public void fly()
    {
        System.out.println(
            "This bird is flying");
    }
}

```

Setelah kelas "Bird" dibuat, jika kita ingin membuat kelas "Pigeon", maka kita cukup mewarisi kelas "Bird" di atas

```
// Creating the Pigeon class which
// extends the bird class
public class Pigeon extends Bird {

    String animal ="Pigeon";
    // Overriding the fly method
    // which makes this pigeon fly
    @Override
    public void fly() //override method
    {
        System.out.println(
            "Pigeon flies!!!");
    }
}
```

Dan jika kita ingin membuat method yang menggunakan atribut dari kelas "Bird", maka kita dapat memanggilnya dengan menggunakan keyword "super"

```
// Creating the Pigeon class which
// extends the bird class
public class Pigeon extends Bird {

    String animal ="Pigeon";
    // Overriding the fly method
    // which makes this pigeon fly
    @Override
    public void fly() //override method
    {
        System.out.println(
            "Pigeon flies!!!");
    }

    //super & this
    public void about() {
        System.out.println(super.animal+" live freely in the sky");
        System.out.println(this.animal+ " can recognise each letter of the human alphabet");
    }
}
```

ENCAPSULATION

Sekarang, kita telah mendefinisikan properti dari kelas "Bird" dan atribut yang dimiliki burung seperti warna, sayap, kaki dapat diinisialisasi dengan membuat objek kelas burung. Namun, jika kita hanya dapat mengubah properti dari kelas "Bird" hanya dengan referensi dari objek, maka atribut kehilangan informasi yang awalnya diinisialisasi.

Misalnya, katakanlah kita awalnya membuat merpati dengan warna abu-abu dengan membuat konstruktor, setiap pengguna dengan instance objek merpati dapat mengubah warna ini menjadi merah atau hitam hanya dengan merujuk atribut dengan kata kunci "this". Untuk menghindari hal ini, kita menyertakan properti ke dalam sebuah metode. Metode ini disebut getter dan setter atribut. Idennya adalah untuk hanya menyertakan inisialisasi dan pengambilan atribut dalam suatu metode alih-alih langsung merujuk atribut secara langsung. Ini juga memberikan keuntungan karena setter memberi kita kendali penuh dalam menetapkan nilai ke atribut dan membantu kita membatasi perubahan yang tidak perlu.

```
// Implementing the bird class
public class Bird {

    // Few properties which
    // define the bird
    String animal = "All Birds";
    String color;
    int legs;

    // Implementing the getters and
    // setters for the color and legs.

    public void setColor(String color)
    {
        this.color = color;
    }

    public String getColor()
    {
        return this.color;
    }

    public void setLegs(int legs)
    {
        this.legs = legs;
    }

    public int getLegs()
    {
        return this.legs;
    }

    // Few operations which the
    // bird performs
    public void eat()
    {
        System.out.println(
            "This bird has eaten");
    }

    public void fly()
    {
        System.out.println(
            "This bird is flying");
    }
}
```

POLYMORPHISM

Sekarang, kita ingin membuat merpati dapat terbang dengan menggunakan method “fly” namun kita ingin mendefinisikan tujuan terbang merpati tersebut dan kita ingin mendefinisikan makanan yang dimakan oleh merpati tersebut. Kita dapat melakukannya dengan menggunakan konsep overload method.

```
// Creating the Pigeon class which
// extends the bird class
public class Pigeon extends Bird {

    String animal = "Pigeon";
    // Overriding the fly method
    // which makes this pigeon fly
    @Override
    public void fly() //override method
    {
        System.out.println(
            "Pigeon fly!!!");
    }

    //overload method to method in this class
    public void fly(String place){
        System.out.println("Pigeon flys to "+ place);
    }

    //overload method to method in parent class
    public void eat(String food)
    {
        System.out.println(
            "Pigeon eats "+food);
    }

    //super & this
    public void about(){
        System.out.println(super.animal+" live freely in the sky");
        System.out.println(this.animal+" can recognise each letter of the human alphabet");
    }
}
```

TESTING

Setelah kita selesai membuat kelas “Bird” dan kelas “Pigeon”, kita perlu membuat sebuah kelas untuk menjalankan atau memanggil kelas-kelas tersebut beserta method-methodnya.

```
//encapsulation, inheritance, & polymorphism testing
public class OOPTest {
    public static void main(String args[]){
        Bird bird = new Bird();
        Pigeon pigeon = new Pigeon();

        bird.eat();
        bird.fly();

        System.out.println("");

        //polymorphism
        pigeon.eat("Seed");
        pigeon.fly();
        pigeon.fly("Sky");

        System.out.println("");

        //Mutators & inheritance
        pigeon.setColor("grey");
        pigeon.setLegs(2);
        System.out.println(
            "this pigeon has "+pigeon.getColor()+" color "
            + "and "+ pigeon.getLegs()+" legs");
        pigeon.eat();

        System.out.println("");
        pigeon.about();
    }
}
```